Cisco SCMS SM Java API Programmer Guide

Version 3.0
OL-7204-02

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

Customer Order Number: DOC-720402=
Text Part Number: OL-7204-02

Printed in the USA on recycled paper containing 10% postconsumer waste.

Cisco SCMS SM Java API *Programmer Guide*

CONTENTS

Preface

This document explains the ***SCMS SM Java API*** and how to install, compile, and run the API.

The ***SCMS SM Java API*** Programmer Guide is used for updating, querying, and configuring the SCMS Subscriber Manager (SM). It consists of two parts, which may be used separately or together without limitation.

- **SM Non-blocking Java API**—A high-performance API with low visibility to errors and other operation results. Supports automatic integrations with OSS/AAA systems.
- **SM Blocking Java API**—A more user-friendly API. Supports user interface applications for accessing and managing the SM.

Note A set of APIs with exactly the same functionality is also available for the C/C++ environment.

Document Revision History

Cisco Service Center Release	Part Number	Publication Date
Release 3.0	OL-7204-02	December, 2005
	Description of Changes	
	Reorganization of documentation. No major changes or new features were added to this revision.	
Release 2.5.7	OL-7204-01	May, 2005

Audience

This guide is for the networking or computer technician responsible for configuring the Subscriber Manager. It is also intended for the operator who manages the SCE platforms.

Organization

This Programmer Guide contains the following topics:

Chapter	Title	Description
Chapter 1	*Getting Started* (on page 1-1)	Describes the platforms on which the Java API can be used, how to install, compile, and start running the Java API component.
Chapter 2	*General API Concepts* (on page 2-1)	Provides a description of the various concepts that are used when working with the SM Java API.
Chapter 3	*Blocking API* (on page 3-1)	Describes the features and operations of the Blocking API and provides code examples.
Chapter 4	*Non-blocking API* (on page 4-1)	Describes the features and operations of the Non-blocking API and provides code examples.
Appendix A	*List of Error Codes* (on page A-1)	Provides a list of error codes that are used in the Java API.

Related Documentation

This API Programmer Guide should be used in conjunction with all of the SCMS Subscriber Manager User, API, and Reference Guides.

Conventions

This document uses the following conventions:

Convention	Description
boldface font	Commands and keywords are in **boldface**.
italic font	Arguments for which you supply values are in i*talics*.
[ ]	Elements in square brackets are optional.
{x | y | z}	Alternative keywords are grouped in braces and separated by vertical bars.
[x | y | z]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	A nonquoted set of characters. Do not use quotation marks around the string, or the string will include the quotation marks.
`screen` font	Terminal sessions and information the system displays are in `screen` font.
`boldface screen` font	Information you must enter is in **`boldface screen`** font.
`italic screen` font	Arguments for which you supply values are in *`italic screen`* font.

→	This pointer highlights an important line of text in an example.
^	The symbol ^ represents the key labeled **Control**—for example, the key combination **^D** in a screen display means hold down the **Control** key while you press the **D** key.
< >	Non printing characters, such as passwords, are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

Notes use the following conventions:

Note Means *reader take note*. Notes contain helpful suggestions or references to materials not contained in this manual.

Cautions use the following conventions:

Caution Means *reader be careful*. You are capable of doing something that might result in equipment damage or loss of data.

Warnings use the following conventions:

Warning Means *reader be warned.* You are capable of doing something that might result in bodily injury.

Obtaining Documentation

The following sections provide sources for obtaining documentation from Cisco Systems.

World Wide Web

You can access the most current Cisco documentation on the World Wide Web at the following sites:

- *http://www.cisco.com*
- *http://www-china.cisco.com*
- *http://www-europe.cisco.com*

Documentation CD-ROM

Cisco documentation and additional literature are available in a CD-ROM package, which ships with your product. The Documentation CD-ROM is updated monthly and may be more current than printed documentation. The CD-ROM package is available as a single unit or as an annual subscription.

Ordering Documentation

Cisco documentation is available in the following ways:

- Registered Cisco Direct Customers can order Cisco Product documentation from the networking Products MarketPlace:

  http://www.cisco.com/cgi-bin/order/order_root.pl

- Registered Cisco.com users can order the Documentation CD-ROM through the online Subscription Store:

  http://www.cisco.com/pcgi-bin/marketplace/welcome.pl

- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco corporate headquarters (California, USA) at 408 526-7208 or, in North America, by calling 800 553-NETS(6387).

Documentation Feedback

If you are reading Cisco product documentation on the World Wide Web, you can submit technical comments electronically. Click **Feedback** in the toolbar and select **Documentation**. After you complete the form, click **Submit** to send it to Cisco.

You can email your comments to bug-doc@cisco.com.

To submit your comments by mail, use the response card behind the front cover of your document, or write to the following address:

Attn Document Resource Connection

Cisco Systems, Inc.

170 West Tasman Drive

San Jose, CA 95134-9883

We appreciate your comments.

Obtaining Technical Assistance

Cisco provides *Cisco.com* (on page ix) as a starting point for all technical assistance. Customers and partners can obtain documentation., troubleshooting tips, and sample configurations from online tools. For Cisco.com registered users, additional troubleshooting tools are available from the TAC website.

Cisco.com

Cisco.com is the foundation of a suite of interactive, networked services that provides immediate, open access to Cisco information and resources at anytime, from anywhere in the world. This highly integrated Internet application is a powerful, easy-to-use tool for doing business with Cisco.

Cisco.com provides a broad range of features and services to help customers and partners streamline business processes and improve productivity. Through Cisco.com, you can find information about Cisco and our networking solutions, services, and programs. In addition, you can resolve technical issues with online technical support, download and test software packages, and order Cisco learning materials and merchandise. Valuable online skill assessment, training, and certification programs are also available.

Customers and partners can self-register on Cisco.com to obtain additional personalized information and services. Registered users can order products, check on the status of an order, access technical support, and view benefits specific to their relationships with Cisco.

To access Cisco.com, go to the following website:

http://www.cisco.com

Technical Assistance Center

The Cisco TAC website is available to all customers who need technical assistance with a Cisco product or technology that is under warranty or covered by a maintenance contract.

Contacting TAC by Using the Cisco TAC Website

If you have a priority level 3 (P3) or priority level 4 (P4) problem, contact TAC by going to the TAC website:

http://www.cisco.com/tac

P3 and P4 level problems are defined as follows:

- P3—Your network is degraded. Network functionality is noticeably impaired, but most business operations continue.
- P4—You need information or assistance on Cisco product capabilities, product installation, or basic product configuration.

In each of the above cases, use the Cisco TAC website to quickly find answers to your questions.

To register for *Cisco.com* (on page ix), go to the following website:

http://tools.cisco.com/RPF/register/register.do

If you cannot resolve your technical issue by using the TAC online resources, Cisco.com registered users can open a case online by using the TAC Case Open tool at the following website:

http://www.cisco.com/tac/caseopen

Contacting TAC by Telephone

If you have a priority level 1 (P1) or priority level 2 (P2) problem, contact TAC by telephone and immediately open a case. To obtain a directory of toll-free numbers for your country, go to the following website:

http://www.cisco.com/warp/public/687/Directory/DirTAC.shtml

P1 and P2 level problems are defined as follows:

- P1—Your production network is down, causing a critical impact to business operations if service is not restored quickly. No workaround is available.
- P2—Your production network is severely degraded, affecting significant aspects of your business operations. No workaround is available.

Disclaimer: The code in this document can be used as a guideline for your site installation. However, any code that has been implemented using this document is not supported by TAC.

CHAPTER 1

Getting Started

This section describes the platforms on which the Java API can be used and how to install, compile, and start running the API.

This chapter contains the following sections:

Introduction

The Java API is used for updating, querying, and configuring the SCMS Subscriber Manager (SM). It consists of two parts, which can be used separately or together without restriction.

- SM Non-blocking Java API—A high-performance API with low visibility to errors and other operation results. It supports automatic integrations with OSS/AAA systems.
- SM Blocking Java API—A more user-friendly API. It supports user interface applications for accessing and managing the SM.

Platforms

The SM Java API was developed and tested on a Windows platform, but it is operable on any platform that supports Java version 1.4.

Installing the Java API

Extracting the Package

The Java API distribution is provided as part of the SCMS SM-LEG distribution file and is located in the *sm_api* directory

The Java SM API is packaged in a UNIX tar file that can be extracted using the UNIX tar utility or most Windows compression utilities.

To install the distribution on a UNIX platform:

Step 1 Extract the SCMS SM-LEG distribution file and locate the Java SM API distribution tar *sm-java-api-dist.tar*.

Step 2 Extract the Java SM API distribution tar and obtain the sm-java-api-vvv.bb.tar:

```
#> tar -xvf sm-java-api-dist.tar
```

Step 3 Extract the Java SM API package tar:

```
#> tar -xvf sm-java-api-vvv.bb.tar
```

To install the distribution on a Windows platform:

Use a zip extractor (such as WinZip).

Note The abbreviations *vvv* and *bb* stand for the Java SM API version and build number.

Package Content

For brevity, <*installdir*> refers to the installation directory *sm-java-api-vvv.bb*.

The *<installdir>/javadoc* folder contains the API JAVADOC documentation.

The *<installdir>/lib* folder contains the *smapi.jar* file, which is the API Executable. It also contains additional jar files necessary for the API operation.

Table 1-1 Layout of Installation Directory

Path	Name	Description
<installdir>		

Path	Name	Description
	README	API readme file
<installdir>/Javadoc		
	index.html	Index of all API specifications
	(API specification files, etc.)	API specification documents
<installdir>/Lib		
	smapi.jar	SM API executable
	asn1rt.jar	Utility jar used by the API
	jdmkrt.jar	Utility jar used by the API
	log4j.jar	Utility jar used by the API
	log4j.properties	Property file needed for the API logging functionalities
	xerces.jar	Utility jar used by the API

Compiling and Running

To compile and run a program that uses the SM Java API, *`smapi.jar`* must be in the CLASSPATH.

For example, if the program source is in *`SMApiProgram.java`*, use the following command line to compile the program:

```
#> javac -classpath smapi.jar SMApiProgram.java
```

After compiling the program, use the following command line to run the program:

```
#> java -cp .;<installdir>/lib/smapi.jar SMApiProgram
```

Subscriber Manager Setup

The API connects to the PRPC server on the SM. For the API to work:

- The SM must be up and running, and reachable from the machine that hosts the API
- The PRPC server must be started.

The PRPC server is a proprietary RPC protocol designed by Cisco. For more information about the PRPC server, see the *SCMS Subscriber Manager User Guide.*

General API Concepts

This section describes the various concepts that are used while working with the SM Java API.

This chapter contains the following sections:

Blocking API/Non-blocking API

This section describes the differences between the Blocking API and the Non-blocking API operations.

Blocking API

In a Blocking API operation, which is the most common, every method returns *after* its operation has been performed.

The SM Blocking Java API provides a wide range of operations. It contains most of the functionality of the Non-blocking API and many functions that are not provided by the Non-blocking API. Reliability and auto-reconnect functionality is not supported in the Blocking API operations.

Non-blocking API

The SM Non-blocking Java API methods return immediately, even before their operation has been completed. The operation results are either returned to an Observer object (Listener) or not returned at all.

The Non-blocking API method is advantageous when the operation is lengthy and involves I/O. Performing the operation in a separate thread allows the calling program to continue doing other tasks and it improves overall system performance.

The SM Non-blocking Java API contains a small number of non-blocking operations. The API supports retrieval of operation results using a result listener.

The SM Non-blocking Java API supports two modes: *reliable* and *non-reliable*. For more information about the reliability modes, see *Reliability Support* (on page 4-1).

API Initialization

To initialize the API:

Step 1 Construct the API using one of its constructors.

Step 2 Perform the API-specific setup operations.

Step 3 Connect the API to the SM.

The three steps above are described in the following sections.

Initialization examples can be found within the code examples sections under each API.

API Construction

Blocking and Non-blocking APIs have two common constructors:

- An empty constructor
- A constructor that accepts a **LEG name** as a parameter

Constructor that Accepts a LEG Name

Set the LEG name if you intend to turn on the SM-LEG failure handling options in the SM. You should read about the LEG software components and SM-LEG failure handling in the *SCMS Subscriber Manager User Guide*.

The LEG name will be used by the SM when recovering from a connection failure. A constant string that identifies the API will be appended to the LEG name as follows:

- For Blocking API: `.B.SM-API.J`
- For Non-blocking API: `.NB.SM-API.J`

Example (Blocking API):

- If the provided LEG name is `my-leg.10.1.12.45-version-1.0`, the actual LEG name will be `my-leg.10.1.12.45-version-1.0.B.SM-API.J`.

If no name is set, the LEG uses the hostname of the machine as the prefix of the name.

For additional information about LEG-SM failure handling, see *Appendix A* of the *SCMS Subscriber Manager User Guide*.

Additional constructors are available for the Non-blocking API. For more information, see *Non-blocking API Construction (on page 4-4)*.

Setup Operations

The setup operations differ for the two APIs. Both APIs support setting a disconnect listener, described in detail in the *DisconnectListener Interface* (on page 2-7) section.

Blocking API Setup

To set up the Blocking API, you need to set an operation timeout value. For more information, see the *Blocking API (on page 3-1) chapter*.

Non-blocking API Setup

To set up the Non-blocking API you are required to set a disconnect listener. For more details, see the *Non-blocking API* (on page 4-1) chapter.

Connecting to the Subscriber Manager

To connect to the SM, use one of the following `connect` methods.

- Use the following method to connect to the SM using the default RPC TCP port (14374):

```
connect(String host)
```

- Use the following method to allow the caller to set the TCP port to which the API connects:

```
connect(String host, int port)
```

For both methods, the `host` parameter can be either an IP address or a reachable hostname.

At any time during the API operation, you can check if the API is connected by using the `isConnected` method.

API Finalization

To free the resources of both server and client, use the `disconnect` method.

It is recommended that you use a `finally` statement in your main class; for example:

```
public static void main(String [] args) throws Exception {
   SMNonBlockingApi smnbapi = new SMNonBlockingApi();
   try {
         …
   } finally {
         smnbapi.disconnect();
   }
}
```

Subscriber Name Format

Most methods of both APIs require the subscriber name as an input parameter. This section lists the formatting rules of a subscriber name.

The subscriber name is *case-sensitive*. It may contain up to 40 characters. The following characters can be used:

Alphanumerics	$ (dollar sign)	. (period or dot)	_ (underscore)
- (minus sign or hyphen)	% (percent sign)	/ (slash)	~ (tilde)
! (exclamation mark)	& (ampersand)	: (colon)	' (apostrophe)
# (number sign)	() (parentheses)	@ (at sign)	

Network ID Mappings

A network ID mapping is a network identifier that the SCE device can relate to a specific subscriber record. A typical example of a network ID mapping (or simply mapping) is an IP address. Currently, the Cisco Service Control solution supports IP address, IP range, and VLAN mappings.

Both Blocking and Non-blocking APIs contain operations that accept mappings as a parameter. Examples are:

- the `addSubscriber` operation (Blocking API)
- the `login` method (Blocking or Non-blocking API)

When passing mappings to an API method, the caller is requested to provide two parameters:

- a `java.lang.String` mapping identifier or array of mapping types
- a short mapping type or array of mapping types

When passing arrays, the `mappingTypes` array must contain either the same number of elements as the `mappings` array, or a single element. If the `mappingTypes` array contains a single element, all mappings have the same type, specified by this single element.

The API supports the following subscriber mapping types:

- IP addresses or IP ranges

- VLAN tags

For additional information, see the *SCMS Subscriber Manager User Guide*.

Specifying IP Address Mapping

The string format of an IP address is the commonly used decimal notation: `IP-Address=[0-255].[0-255].[0-255].[0-255].`

Examples:

- `216.109.118.66`
- The mapping type of an IP address is provided in the interface `com.pcube.management.api.SMApiConstants`:
  - `com.pcube.management.api.SMApiConstants.MAPPING_TYPE_IP` specifies a single IP mapping that matches the mapping identifier with the same index in the mapping identifier array.
  - `com.pcube.management.api.SMApiConstants.ALL_IP_MAPPINGS` specifies that all the entries in the mapping identifiers array are IP mappings.

Specifying IP Range Mapping

The string format of an IP range is an IP address in decimal notation and a decimal specifying the number of **1**s in a bit mask: `IP-Range=[0-255].[0-255].[0-255].[0-255]/[0-32]`.

Examples:

- `10.1.1.10/32` is an IP range with a full mask, that is, a regular IP address.
- `10.1.1.0/24` is an IP range with a 24-bit mask, that is, all the addresses ranging between `10.1.1.0` and `10.1.1.255`.

Note The mapping type of an IP Range is identical to the mapping type of the IP address.

Specifying VLAN Tag Mapping

The string format for VLAN tag mapping is `VLAN-tag = 0-4095`.

The value is a decimal in the specified range.

The mapping type is also provided in interface `com.pcube.management.api.SMApiConstants`:

- `com.pcube.management.api.SMApiConstants.MAPPING_TYPE_VLAN` specifies a single VLAN mapping that matches the mapping identifier with the same index in the mapping identifier array.
- `com.pcube.management.api.SMApiConstants.ALL_VLAN_MAPPINGS` specifies that all the entries in the mapping identifiers array are VLAN mappings.

Subscriber Domains

The domain concept is explained in detail in the *SCMS Subscriber Manager User Guide*. Roughly, a domain is an identifier that tells the SM which SCE devices should be updated with the subscriber record.

A domain name is of type `String`. During system installation, the network administration determines the system domain names, which therefore vary between installations. The APIs include methods that specify to which domain a subscriber belongs and allow queries about the system's domain names. If an API operation specifies a domain name that does not exist in the SM domain repository, it is considered an error and an `RpcErrorException` will be returned.

Subscriber Properties

Several operations manipulate subscriber properties. A subscriber property is a key-value pair that affects the way the SCE analyzes and reacts to network traffic generated by the subscriber.

More information about properties can be found in the *SCMS Subscriber Manager User Guide* and in the *Service Control Application for Broadband User Guide*. The application user guide provides application-specific information; it lists the subscriber properties that exist in the application running on your system, the allowed value set, and the significance of each property value.

To format subscriber properties for Java API operations, use the String arrays `propertyKeys` and `propertyValues`.

Note The arrays must be of the *same length*, and NULL entries are forbidden. Each key in the keys array has a matching entry in the values array; the value for `propertyKeys[j]` resides in `propertyValues[j]`.The mapping type of an IP Range is identical to the mapping type of the IP address.

Example:

- If the property keys array is `{“name”,“color”,“shape”}` and the property values array is `{“john”,“red”,“circle”}`, the properties will be `name=john, color=red, shape=circle`.

Custom Properties

Some operations manipulate custom properties. Custom properties are similar to subscriber properties, but do not affect how the SCE analyzes and manipulates the subscriber's traffic. The application management modules use custom properties to store additional information for each subscriber.

To format custom properties, use the `String` arrays `customPropertyKeys` and `customPropertyValues`, the same as in formatting *Subscriber Properties* (on page 2-6).

DisconnectListener Interface

Both APIs (Blocking and Non-blocking) allow setting a disconnect listener. The disconnect listener is an interface with a single method:

```
public interface DisconnectListener {

   /**
    * called when the connection with the server is down.
    */
   public void connectionIsDown();
}
```

Implement this interface to be notified when the API is disconnected from the SM.

To set a disconnect listener, use the `setDisconnectListener` method.

Example

The following example is a simple implementation of a disconnect listener that prints a message to `stdout` and exits.

```
import com.pcube.management.framework.rpc.DisconnectListener;

public class MyDisconnectListener implements DisconnectListener {

   public void connectionIsDown(){
         System.out.println("Message: connection is down.");
         System.exit(0);
   }
}
```

Exceptions

All functional errors of the SM Java API are provided by the same Java class, `com.pcube.management.framework.rpc.RpcErrorException`, which is contrary to the normal Java usage. This approach was chosen because of the "cross-language" nature of the SM API. It allows all SM API implementations (Java, C, C++) to look and feel the same.

Each exception provides the following information:

- A unique error code (`long`)
- An informative message (`java.lang.String`)
- A server-side stack trace (`java.lang.String`)

The error code can be interpreted using `com.pcube.management.api.SMApiConstants`. See the *List of Error Codes* (on page A-1) for more details about error codes and their significance.

Note Several types of errors can occur **only** when the Blocking API is used. These are operational errors related to operation-timeout handling. They are described in detail in the *Blocking API* (on page 3-1) chapter.

Practical Tips

When implementing the code that integrates the API with your application, you should consider the following practical tips:

- Connect to the SM once and maintain an open API connection to the SM at all times, using the API many times. Establishing a connection is a timely procedure, which allocates resources on the SM side and the API client side.
- Share the API connection between your threads. It is better to have one connection per LEG. Multiple connections require more resources on the SM and client side.
- Do not implement synchronization of the calls to the API. The client automatically synchronizes calls to the API.
- It is recommended to place the API clients (LEGs) in the same order of the SM machine processor number.
- If the LEG application has bursts of logon operations, enlarge the internal buffer size accordingly to hold these bursts (Non-Blocking flavor).
- During the integration, set the SM *logon_logging_enabled* configuration parameter to view the API operations in the SM log to troubleshoot the integration, if any problems arise.
- Use the debug mode for the LEG application that logs/prints the return values of the non-blocking operations.
- Use the automatic reconnect feature to improve the resiliency of the connection to the SM.
- In cluster setups, connect the API using the virtual IP address of the cluster and not the management IP address of one of the machines.

CHAPTER 3

Blocking API

This chapter introduces the Reply Timeout, a feature unique to the Blocking API. The rest of the chapter lists all operations of the Blocking API, and provides code examples.

Note If you only need to develop an *automatic integration*, skip this chapter and go directly to the *Non-blocking API* (on page 4-1) chapter.

This chapter contains the following sections:

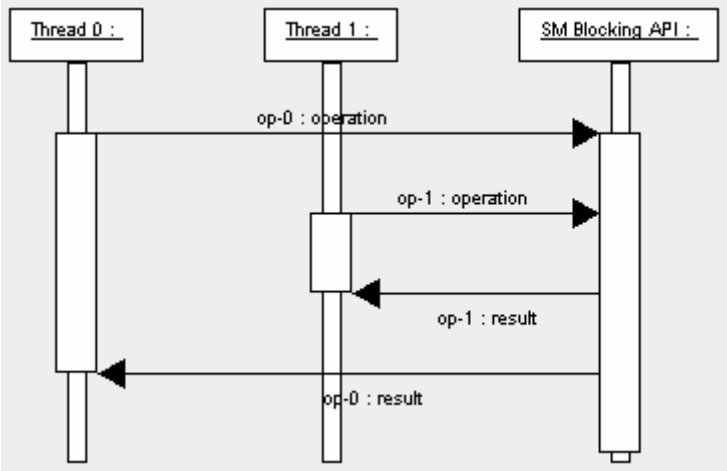
Multi-threading Support

The Blocking API supports unlimited number of threads calling its methods simultaneously.

Note In a multi-threaded scenario for the Blocking API, the order of invocation is **not** guaranteed.

Example:

Thread-0 calls operation-0 at time-0, and thread-1 calls operation-1 at time-1, where time-1 is later than time-0. In this example, it is possible that operation-1 may be performed **before** operation-0, as shown in the following diagram (the vertical scale is time):



The SM allocates five threads to handle each API instance. It is recommended to develop a multi-threaded application that uses the API with a number of threads in the order of the five threads. Implementing with more threads might result in longer delays for the calling threads.

ReplyTimeout and OperationTimeout Exception

A blocking operation returns only when the operation result has been retrieved from the SM. If a networking malfunction or other error prevents the operation result from being retrieved, the caller will wait indefinitely. The SM API provides means of working around this situation.

The reply timeout feature (the `setReplyTimeout` method) lets the caller set a timeout. It will fire a `com.pcube.management.framework.rpc.OperationTimeoutException` when a reply does not return within the timeout period.

Calling the `setReplyTimeout` method with a `long` value sets a reply timeout. The reply timeout is interpreted in milliseconds. A zero value indicates that the operation should wait (freeze, hang) until a result arrives - or indefinitely, if no result arrives.

There is an alternate way of releasing a method call that is blocking the caller, who is waiting for a result to arrive: Call the `interrupt` method of the calling thread: a `java.lang.InterruptedException` will then be returned to the caller.

Blocking API Methods

This section lists the methods of the Blocking API. The syntax of each method is followed by a description of its input parameters and its return values.

The Blocking API is a superset of the Non-blocking API. Except for differences in return values and result handling, identical operations in both APIs have the same functions and syntax structure.

All the methods throw a `java.lang.IllegalStateException` when called before a connection with the SM is established.

The Blocking API methods can be classified into the following categories:

- **Dynamic IP and property allocation—**For using the SM API for integration with an AAA system, the following methods are relevant. These methods are not designed to add or remove subscribers from the database, but to modify dynamic parameters (such as IP addresses) of existing subscribers:
  - *login* (on page 3-5)
  - *logoutByName* (on page 3-8)
  - *logoutByNameFromDomain* (on page 3-10)
  - *logoutByMapping* (on page 3-12)
  - *loginCable* (on page 3-13)
  - *logoutCable* (on page 3-15)
- **Static/Manual Subscriber configuration—**For example, for GUI usage, the following methods are relevant:
  - *addSubscriber* (on page 3-16)
  - *removeSubscriber* (on page 3-19)
  - *removeAllSubscribers* (on page 3-20)
  - *setPropertiesToDefault* (on page 3-34)
  - *removeCustomProperties* (on page 3-35)
- For simple read-only operations, performed independently on the subscriber awareness mode, the following methods are relevant:
  - *getNumberOfSubscribers* (on page 3-21)
  - *getNumberOfSubscribersInDomain* (on page 3-22)
  - *getSubscriber* (on page 3-23)
  - *subscriberExists* (on page 3-25)
  - *subscriberLoggedIn* (on page 3-26)
  - *getSubscriberNameByMapping* (on page 3-27)
  - *getSubscriberNames* (on page 3-28)
  - *getSubscriberNamesInDomain* (on page 3-30)

- *getSubscriberNamesWithPrefix* (on page 3-31)
- *getSubscriberNamesWithSuffix* (on page 3-32)
- *getDomains* (on page 3-33)

It is possible to mix methods from different categories in a single application. The classification is presented for clarification purposes only.

login

Syntax

```
public void login(String subscriberName,
                  String[] mappings,
                  short[] mappingTypes,
                  String[] propertyKeys,
                  String[] propertyValues,
                  String domain,
                  boolean isMappingAdditive,
                  int autoLogoutTime)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

The `login` method adds or modifies a domain, mappings and possibly properties of a subscriber who already exists in the SM database. Login can be called with partial data; for example, with only mappings or only properties provided and NULL put in the unchanged fields.

If another subscriber with the same (or colliding) mappings already exists in the same domain, the colliding mappings will be removed from the other subscriber and assigned to the new subscriber.

If the subscriber does not exist in the SM database, it will be created with the data provided.

Parameters

`subscriberName:` See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`mappings:` See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.
If no mappings are specified, and the `isMappingAdditive` flag is TRUE, the previous mappings will be retained. If no such mappings exist, the operation will fail.

`mappingTypes:` See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`propertyKeys:` See explanation of property keys and values in the *General API Concepts* (on page 2-1) chapter.

`propertyValues:` See explanation of property keys and values in the *General API Concepts* (on page 2-1) chapter.

`domain:` See explanation of domains in the *General API Concepts* (on page 2-1) chapter.
If domain is NULL, but the subscriber already has a domain, the existing domain will be retained.

`isMappingAdditive:`

- `TRUE`: adds the mappings provided by this call to the subscriber record.
- `FALSE`: overrides the mappings provided by this call with mappings that already exist in the subscriber record.

`autoLogoutTime:`

Applies only to mappings provided as arguments to this method.

- Positive value (N): automatically logs out the mappings (similar to a logout method being called) after N seconds.
- 0 value: maintains current expiration time for the given mappings.
- Negative value: disables any expiration time that might have been set for the mappings given.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DATABASE_EXCEPTION
- ERROR_CODE_UNKNOWN

  This error can be caused by the following:

  - NULL value for domain parameter for the subscriber that does not exist or does not have a domain
  - Invalid values for propertyValues parameter

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

Return Value

None.

Examples

To add the IP address 192.168.12.5 to an existing subscriber named *john* without affecting existing mappings:

```
login(
   “john”,                                  // subscriber name
   new String[]{“192.168.12.5”},
   SMApiConstants.ALL_IP_MAPPINGS,
   null, null,
   ”subscribers”,                  // domain
   true,                                    // isMappingAdditive is true
   -1);                                // autoLogoutTime set to infinite
```

To add the IP address 192.168.12.5 overriding previous mappings:

```
login(
   “john”,                                  // subscriber name
   new String[]{“192.168.12.5”},
   SMApiConstants.ALL_IP_MAPPINGS,
   null, null,
   ”subscribers”,                  // domain
   false,                                   // isMappingAdditive is false
   -1);                                // autoLogoutTime set to infinite
```

To extend the auto logout time of 192.168.12.5 that was previously assigned to *john*:

```
login(
   “john”,
                        //the previously assigned IP
   new String[]{“192.168.12.5”},
   SMApiConstants.ALL_IP_MAPPINGS,
   null, null,
   “subscribers”,                           // domain
   false,                                   // isMappingAdditive
   300);                               // autoLogoutTime set to 300 seconds
```

To modify a dynamic property of *john* (e.g. package ID):

```
login(
   “john”,
   null, null,
   new String[]{“packageId”},      // property key
   new String[]{“10”},                    // property value
   “subscribers”,                  // domain
   false, -1);
```

To add the IP address 192.168.12.5 to an existing subscriber named *john* without affecting existing mappings and modify a dynamic property of *john* (e.g. package ID):

```
login(
   “john”,
   new String[]{“192.168.12.5”},
   SMApiConstants.ALL_IP_MAPPINGS,
   new String[]{“packageId”},      // property key
   new String[]{“10”},                    // property value
   “subscribers”,                  // domain
   true,                                    // isMappingAdditive is set to
true
   -1);
```

logoutByName

Syntax

```
public boolean logoutByName(String subscriberName,
                            String[] mappings,
                            short[] mappingTypes)
throws InterruptedException, OperationTimeoutException,
RpcErrorException
```

Description

Locates the subscriber in the database and removes mappings from it. If the subscriber does not exist it does nothing.

Parameters

`subscriberName:` See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`mappings:` See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.
If no mappings are specified, all the subscriber mappings will be removed.

`mappingTypes:` See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

Return Value

- TRUE—if the subscriber was found and the subscriber's mappings were removed from the subscriber database.
- FALSE—if the subscriber was not found in the subscriber database.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description error codes, see *Appendix A - List of Error Codes* (on page A-1).

Examples

To remove IP address 192.168.12.5 of subscriber *john*:

```
boolean isExist = logoutByName(
   “john”,
   new String[]{“192.168.12.5”},
   SMApiConstants.ALL_IP_MAPPINGS);
```

To remove all IP addresses of subscriber *john*:

```
boolean isExist = logoutByName(“john”, null, null);
```

logoutByNameFromDomain

Syntax

```
public boolean logoutByNameFromDomain(String subscriberName,
                                      String[] mappings,
                                      short[] mappingTypes,
                                      String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Similar to `logoutByName`, but also lets the caller provide the name of the domain to which the subscriber belongs. When the subscriber domain is known, use this method to get improved performance.

Parameters

`subscriberName:` See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts (on page 2-1)* chapter.

`mappings:` See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts (on page 2-1)* chapter.
If no mappings are specified, all the subscriber mappings will be removed.

`mappingTypes:` See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts (on page 2-1)* chapter.

`domain:` See explanation of domains in the *General API Concepts* (on page 2-1) chapter.
The operation will fail if *either* of the following conditions exists:

- The domain is null, but the subscriber exists in the database and belongs to a domain.
- The domain specified is incorrect.

Return Value

- TRUE—if the subscriber was found and removed from the subscriber database
- FALSE—if the subscriber was not found in the subscriber database.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

Example

To remove IP address 192.168.12.5 of subscriber *john* from domain *subscribers*:

```
boolean isExist = logoutByNameFromDomain(
   “john”,
   new String[]{“192.168.12.5”},
   SMApiConstants.ALL_IP_MAPPINGS,
   “subscribers”);
```

To remove all IP addresses of subscriber *john* from domain *subscribers*:

```
boolean isExist = logoutByNameFromDomain(
   “john”,
   null,
   null,
   “subscribers”);
```

logoutByMapping

Syntax

```
public boolean logoutByMapping(String mapping,
                               short mappingType,
                               String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Locates a subscriber based on domain and mapping, and removes the mapping (the subscriber stays in the database).

Parameters

`mapping`: See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`mappingType`: See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`domain`: See description in `logoutByNameFromDomain` (on page 3-10) operation.

Return Value

- TRUE—if the subscriber was found and removed from the subscriber database.
- FALSE—if the subscriber was not found in the subscriber database.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

Example

To remove IP address 192.168.12.5 from domain *subscribers*:

```
boolean isExist = logoutByMapping(
   “192.168.12.5”,
   SMApiConstants. MAPPING_TYPE_IP,
   “subscribers”);
```

loginCable

Syntax

```
public void loginCable(String CPE,
                       String CM,
                       String IP,
                       int lease,
                       String domain,
                       String[] propertyKeys,
                       String[] propertyValues)
throws InterruptedException, OperationTimeoutException,
RpcErrorException
```

Description

A login method adapted for the cable environment (calls the cable support module in the SM). This method is designed to log in CPEs and CMs to the SM. To log in a CPE, specify its CM MAC in the CM argument and the CPE MAC in the CPE argument. To log in a CM, specify the CM MAC address in both CPE and CM arguments. Note that the login of a CPE whose CM does not exist in the SM database will be ignored: the CM has to exist in the database, either by import or by a CM login operation. For additional information, see the *Cable Environment Appendix* of the *SCMS Subscriber Manager User Guide*.

Note The name of the CPE in the SM database is the concatenation of the CPE and CM values with **two** underscore ['_'] characters between them. The caller must make sure that the lengths of CPE and CM add up to no more than **38** characters.

Parameters

`CPE:` A unique identifier of the CPE (usually a MAC address)

`CM:` A unique identifier of the cable modem (usually a MAC address)

`IP:` the CPE IP address

`lease:` the CPE lease time

`domain:` See explanation of *domains* ("Subscriber Domains" on page 2-6) in the *General API Concepts* (on page 2-1) chapter.
The domain will usually be CMTS IP.

Note Domain aliases must be set on the SM in order for the CMTS IP to be correctly interpreted as a domain name. For information regarding aliases configuration read *Configuring Domains* section of *SCMS Subscriber Manager User Guide*.

`propertyKeys:` See explanation of *property keys* ("Subscriber Properties" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.
If the CPE is provided with partial or no application properties, the values for the missing application properties will be copied from the application properties of the CM to which this CPE belongs. Each CM application property thus serves as a default for the CPE under it.

`propertyValues:` See explanation of *property keys* ("Subscriber Properties" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

Return Value

None

RPC Exception Error Codes

None

Examples

To add the IP address 192.168.12.5 to a CM called *CM1* with 2 hours lease time:

```
loginCable(
   “CM1”,
   “CM1”,
   “192.168.12.5”,
   7200,                    // lease time in seconds
   “subscribers”, null, null);
```

To add the IP address 192.168.12.50 to a CPE called *CPE1* which is behind *CM1* with lease time of 1 hours*:*

```
loginCable(
   “CPE1”,
   “CM1”,
   “192.168.12.50”,
   3600,                    // lease time in seconds
   “subscribers”, null, null);
```

logoutCable

Syntax

```
public boolean logoutCable(String CPE,
                           String CM,
                           String IP,
                           String domain)
```

Description

Indicates a logout (CPE becoming offline) event to the SM cable support module.

Parameters

`CPE:` See description in the `loginCable` (on page 3-13) method.

`CM:` See description in the `loginCable` (on page 3-13) method.

`IP:` See description in the `loginCable` (on page 3-13) method.

`domain:` See description in the `loginCable (on page 3-13)` method.

Return Value

- TRUE—if the CPE was found and removed from the subscriber database.
- FALSE—if the CPE was not found in the subscriber database.

RPC Exception Error Codes

None

Examples

To remove the IP address 192.168.12.5 from *CPE1* which is behind *CM1*:

```
boolean isExist = logoutCable(
   “CPE1”,
   “CM1”,
   “192.168.12.5”,
   “subscribers”);
```

addSubscriber

Syntax

```
public void addSubscriber(String subscriberName,
                          String[] mappings,
                          short[] mappingTypes,
                          String[] propertyKeys,
                          String[] propertyValues,
                          String[] customPropertyKeys,
                          String[] customPropertyValues,
                          String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Creates a new subscriber record according and adds the record to the SM database.

If a subscriber by this name already exists, it will be removed before the new one is added. In contrast to `login`, which modifies fields and leaves unspecified fields unchanged, `addSubscriber` sets the subscriber exactly as specified by the parameters passed to it.

Note It is recommended to call `login` method for existing subscribers, instead of `addSubscriber`. Dynamic mappings and properties should be set by using `login`. Static mappings and properties should be set at the first time the subscriber is created by using `addSubscriber`.

Note With `addSubscriber`, the auto-logout feature is always disabled. To enable auto-logout, use `login`.

Example

Subscriber *AB*, already set up in the subscriber database, has a single IP mapping: *IP1*.

If an `addSubscriber` operation for *AB* is called with no mappings specified (NULL in both the *mappings* and *mappingTypes* fields), *AB* will be left with no mappings.

However, calling the `login` operation with these NULL-value parameters will not change *AB*'s mappings; *AB* will be left with its previous IP mapping: *IP1*.

Parameters

`subscriberName:` See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`mappings:` See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`mappingTypes:` See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`propertyKeys:` See explanation of *property keys* ("Subscriber Properties" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

`propertyValues:` See explanation of *property keys* ("Subscriber Properties" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

`customPropertyKeys:` See explanation of *custom property keys* ("Custom Properties" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

`customPropertyValues:` See explanation of *custom property keys* ("Custom Properties" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

`domain:` See explanation of *domains* ("Subscriber Domains" on page 2-6) in the *General API Concepts* (on page 2-1) chapter.
A NULL value indicates that the subscriber is domain-less.

Return Value

None

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_SUBSCRIBER_ALREADY_EXISTS
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_UNKNOWN

  This error code may indicate invalid values that were supplied for `propertyValues` parameter.
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

Examples

To add a new subscriber, *john*, with some custom properties:

```
addSubscriber(   “john”,
   null, null,              // dynamic mappings will be set by login
   null, null               // dynamic properties will be set by login
   new String[]{                          // custom property keys
               “work phone”,
               “home phone”},
   new String[]{                          // custom property values
               “6543212”
               “5059927”},
   “subscribers”);                 // default domain
```

removeSubscriber

Syntax

```
public boolean removeSubscriber(String subscriberName)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Removes a subscriber completely from the SM database.

Parameters

`subscriberName`: See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

Return Value

- TRUE—if the subscriber was found in the database and successfully removed.
- FALSE—if the conditions for TRUE were not met: the subscriber was not found in the database, or the subscriber was found but was not successfully removed.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

Example

To remove subscriber *john* entirely from the database:

```
boolean isExist = removeSubscriber(“john”);
```

removeAllSubscribers

Syntax

```
public void removeAllSubscribers()
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Removes all subscribers from the SM, leaving the database with no subscribers.

Note This method may take time to execute. To avoid operation timeout exceptions, set a high operation timeout (up to 5 minutes) before calling this method.

Return Value

None.

RPC Exception Error Codes

None.

getNumberOfSubscribers

Syntax

```
public int getNumberOfSubscribers()
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Retrieves the total number of subscribers in the SM database.

Return Value

The number of subscribers in the SM.

RPC Exception Error Codes

None

getNumberOfSubscribersInDomain

Syntax

```
public int getNumberOfSubscribersInDomain(String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Retrieves the number of subscribers in a subscriber domain.

Parameters

`domain:` A name of a subscriber domain that exists in the SM's domain repository.

Return Value

The number of subscribers in the domain provided.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE _DOMAIN_NOT_FOUND

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

getSubscriber

Syntax

```
public Object[] getSubscriber(String subscriberName)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Retrieves subscriber record. Each field is formatted as an integer, string, or string array, as described below in the Return Value section for this method.
If the subscriber does not exist in the SM database, an exception will be returned.

Parameters

`subscriberName:` See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

Return Value

An Object Array with nine elements. The index values are listed in the following table. No array element is NULL.

Index 0	subscriber name (`java.lang.String`)
Index 1	array of mappings (`java.lang.String[]`)
Index 2	array of mapping types (`short[]`)
Index 3	domain name (`java.lang.String`)
Index 4	array of property names (`java.lang.String[]`)
Index 5	array of property values (`java.lang.String[]`)
Index 6	array of custom property names (`java.lang.String[]`)
Index 7	array of custom property values (`java.lang.String[]`)
Index 8	auto-logout time, as seconds from now, or -1 if not set (`long[]`)

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

Example

To retrieve the subscriber record of *john*:

```
Object[] subRecord = getSubscriber(“john”);
String[] mappings = (String[])subRecord[1]
short[] mappingTypes = {short[])subRecord[2];
String domainName = (String)subRecord[3];
String[] propertyNames = (String[])subRecord[4];
String[] propertyValues = (String[])subRecord[5];
String[] customPropertyName = (String[])subRecord[6];
String[] customPropertyValues = (String[])subRecord[7];
long[] autoLogoutTime = (long[])subRecord[8];
```

subscriberExists

Syntax

```
public boolean subscriberExists(String subscriberName)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Verifies that a subscriber exists in the SM database.

Parameters

`subscriberName`: See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter

Return Value

- TRUE—if the subscriber was found in the SM database.
- FALSE—if the subscriber could not be found.

RPC Exception Error Codes

None

subscriberLoggedIn

Syntax

```
public boolean subscriberLoggedIn(String subscriberName)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Checks whether a subscriber that already exists in the SM database is logged in, that is, if the subscriber also exists in some SCE database.

When the SM is configured to work in *Pull mode*, a TRUE value returned by this method does not guarantee that the subscriber actually exists in some SCE database, but rather the subscriber is available to be pulled by an SCE, if needed.

If the subscriber does not exist in the SM database, an exception will be returned.

Parameters

`subscriberName:` See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

Return Value

- TRUE—if the subscriber is logged in.
- FALSE—if the subscriber is not logged in.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

getSubscriberNameByMapping

Syntax

```
public String getSubscriberNameByMapping(String mapping,
                                         short mappingType,
                                         String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Finds a subscriber name according to a mapping and a domain.

Parameters

`mapping:` See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`mappingType:` See explanation of mappings and *mapping types* ("Network ID Mappings" on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`domain:` The name of the domain to which the subscriber belongs to. The operation will fail if *either* of the following conditions exists:

- The domain is null, but the subscriber exists in the database and belongs to a domain.
- The specified domain is incorrect.

Return Value

- Subscriber name—if a subscriber record was found.
- NULL—if no subscriber record could be found.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

getSubscriberNames

Syntax

```
public String[] getSubscriberNames(String lastBulkEnd,
                                   int numOfSubscribers)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Gets a bulk of subscriber names from the SM database, starting with `lastBulkEnd` followed by the next `numOfSubscribers` subscribers (in alphabetical order).

If `lastBulkEnd` is NULL, the (alphabetically) first subscriber name that exists in the SM database will be used.

Note There is **no** guarantee that the total number of subscribers (in all bulks) will equal the value returned from `getNumOfSubscribers` at any time. This may differ, for example, if some subscribers are added or removed while bulks are being retrieved.

Parameters

`lastBulkEnd:` Last subscriber name from last bulk. Use NULL to start with the first (alphabetic) subscriber.

`numOfSubscribers:` Limit on number of subscribers that will be returned. If this value is higher than the SM limit (1000), the SM limit will be used.

Note Providing values higher than 500 to this parameter is **not** recommended.

Return Value

An array of subscriber names ordered alphabetically.

The method will return as many subscribers as are found in the SM database, starting at the requested subscriber. The array size is limited by the minimum between `numOfSubscribers` and the SM limit (1000).

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

Example

```
boolean hasMoreSubscribers;
String lastBulkEnd = null;
int bulkSize = 100;

do {
     String[] subscribers = smApi.getSubscriberNames(lastBulkEnd,
bulkSize);

     hasMoreSubscribers = false;
     if (subscribers != null) {
        for (int i = 0; i < subscribers.length; i++) {
              // do something with subscribers[i]
        }
        if (subscribers.length == bulkSize) {
             hasMoreSubscribers = true;
             lastBulkEnd = subscribers[bulkSize - 1];
        }
     }

} while (hasMoreSubscribers);
```

getSubscriberNamesInDomain

Syntax

```
public String[] getSubscriberNamesInDomain(String lastBulkEnd,
                                           int numOfSubscribers,
                                           String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Gets subscribers in the SM database that are associated with the specified domain.

The semantics of this operation are the same as the semantics of the `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation.

Parameters

`lastBulkEnd:` See description in `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation.

`numOfSubscribers:` See description in `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation.

`domain:` The name of a subscriber domain that exists in the SM domain repository.

Return Value

An alphabetically ordered array of subscriber names that belong to the domain provided.

See also the documentation of the Return Value section of the `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _DOMAIN_NOT_FOUND
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

getSubscriberNamesWithPrefix

Syntax

```
public String[] getSubscriberNamesWithPrefix(String lastBulkEnd,
                                             int numOfSubscribers,
                                             String prefix)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Gets subscribers in the SM database whose name begins with a specified prefix.

The semantics of this operation are the same as the semantics of the `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation.

Parameters

`lastBulkEnd:` See description in `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation.

`numOfSubscribers:` See description in `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation.

`prefix:` A case-sensitive string that marks the prefix of the required subscriber names.

Return Value

An alphabetically ordered array of subscriber names that start with the prefix required.

See also the documentation of the Return Value section of the `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

getSubscriberNamesWithSuffix

Syntax

```
public String[] getSubscriberNamesWithSuffix(String lastBulkEnd,
                                             int numOfSubscribers,
                                             String suffix)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Gets subscribers in the SM database whose names end with the specified suffix.

The semantics of this operation are the same as the semantics of the `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation.

Parameters

`lastBulkEnd:` See description in `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation

`numOfSubscribers:` See description in `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation.

`suffix`- A case-sensitive string that marks the suffix of the required subscriber names.

Return Value

An alphabetically ordered array of subscriber names that end with the suffix required.

See also the documentation of the Return Value section of the `getSusbcriberNames` ("getSubscriberNames" on page 3-28) operation.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

getDomains

Syntax

```
public String[] getDomains()
throws InterruptedException, OperationTimeoutException,
RpcErrorException
```

Description

Provides the list of current subscriber domains in the SM domain repository.

Return Value

A complete list of subscriber domain names in the SM.

RPC Exception Error Codes

None

setPropertiesToDefault

Syntax

```
public void setPropertiesToDefault(String subscriberName,
                                   String[] properties)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Resets the specified application properties of a subscriber. If an application is installed, the relevant application properties will be set to the default value of the properties according to the currently loaded application information. If an application is not installed, a `java.lang.IllegalStateException` will be returned.

Parameters

`subscriberName:` See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`properties:` See explanation of *property keys* ("Subscriber Properties" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

Return Value

None.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

removeCustomProperties

Syntax

```
public void removeCustomProperties(String subscriberName,
                                   String[] customProperties)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

Description

Resets the specified custom properties of a subscriber.

Parameters

`subscriberName:` See explanation of *subscriber name format* (on page 2-4) in the *General API Concepts* (on page 2-1) chapter.

`CustomProperties:` See explanation of *custom property keys* ("Custom Properties" on page 2-6) and values in the *General API Concepts* (on page 2-1) chapter.

Return Value

None.

RPC Exception Error Codes

The following list of error codes may be returned by this method:

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

For a description of error codes, see *Appendix A - List of Error Codes* (on page A-1).

Blocking API Code Examples

This section gives two code examples:

- Getting number of subscribers
- Adding subscriber, printing subscriber information, removing subscriber

Getting Number of Subscribers

The following example prints to `stdout` the total number of subscribers in the SM database and the number of subscribers in each subscriber domain.

```
package blocking;

import com.pcube.management.api.SMBlockingApi;

public class PrintInfo {
    public static void main (String args[]) throws Exception {
        SMBlockingApi bapi = new SMBlockingApi();
        try {
            //initiation
            bapi.setReplyTimeout(300000);  //set timeout for 5 minutes
            bapi.connect(args[0]);         // connect to the SM

            //operations
            String[] domains=bapi.getDomains();
            int totalSubscribers=bapi.getNumberOfSubscribers();
            System.out.println(
                "number of susbcribers in the database:\t\t "+
                totalSubscribers);
            for (int i=0; i<domains.length; i++) {
                int numberOfSusbcribersInDomain=
                    bapi.getNumberOfSubscribersInDomain(domains[i]);
                System.out.println(
                    "number of susbcribers domain "+domains[i]+
                    ":\t\t "+numberOfSusbcribersInDomain);
            }
        } finally {
            //finalization
            bapi.disconnect();
        }
    }
}
```

Adding a Subscriber, Printing Information, and Removing a Subscriber

The following program adds a subscriber to the subscriber database, then gets its information and prints it to **stdout**, and finally removes the subscriber from the subscriber database.

```
package blocking;

import com.pcube.management.api.SMBlockingApi;
import com.pcube.management.api.SMApiConstants;

public class AddPrintRemove {
    public static void main (String args[]) throws Exception {
        checkArguments(args);
        SMBlockingApi bapi = new SMBlockingApi();
        try {
            //initiation
            bapi.setReplyTimeout(10000);   //set timeout for 10 seconds
            bapi.connect(args[0]);         // connect to the SM
            //add subscriber
            System.out.println("+ adding subscriber to SM");
            bapi.addSubscriber(
                args[1],   //name
                new String[]{args[2]},   //mapping`
                SMApiConstants.ALL_IP_MAPPINGS,
                new String[]{args[3]}, //property key
                new String[]{args[4]}, //property value
                new String[]{"custom-key"}, //custom property key
                new String[]{"custom-value"}, //custom property value
                args[5]);              //domain
            //Print subscriber
            System.out.println("+ Printing subscriber");
            Object[] subfields = bapi.getSubscriber(args[1]);
            System.out.println("\tname:\t\t"+subfields[0]);
            System.out.println("\tmapping:\t"+
                               ((String[])subfields[1])[0]);
            System.out.println("\tdomain:\t\t"+subfields[3]);
            System.out.println("\tautologout:\t"+subfields[8]);
            //Remove subscriber
            System.out.println("+ removing subscriber from SM");
            bapi.removeSubscriber(args[1]);
        } finally {
            //finalization
            bapi.disconnect();
        }
    }

    static void checkArguments(String[] args)  throws Exception{
        if (args.length != 6) {
            System.err.println(
                "usage: java AddPrintRemove <SM-address>"+
                " <subscriber-name> <IP mapping> <property-key>"+
                " <property-value> <domain>");
            System.exit(1);
        }
    }
}
```

Non-blocking API

This chapter introduces features unique to the Non-blocking API. It lists all methods of the Non-blocking API and ends with code examples.

This chapter contains the following sections:

Reliability Support

The Non-blocking API can work in two different modes, *reliable* and *non-reliable*, as described below. When the mode is not specified, the default is *reliable mode*.

Reliable Mode

In reliable mode, the API ensures that no requests to the SM are lost. The API maintains an internal storage for all API requests that are sent to the SM. Only after a reply from the SM is received, is the request considered **committed** and the API can remove the request from its internal storage. In case of connection failure between the API and the SM, the API accumulates all requests in its internal storage until the connection to the SM is established. On reconnection, the API resends all **non-committed** requests to the SM, so that no requests are lost.

Note In reliable mode, the order of resending requests is **guaranteed**. The API resends the requests in the same chronological order that they were called.

Non-reliable Mode

In non-reliable mode, the API does not ensure that requests sent to the SM are executed. Also, all requests that are sent by the API when connection to the SM is down will be lost unless some external reliability mechanism is implemented.

Auto-reconnect Support

The Non-blocking API supports auto-reconnection to the SM in case of connection failure. When this option is activated, the API can determine when the connection to the SM is lost. When the connection is lost, the API activates a reconnection task that tries to reconnect to the SM until it is successful.

Note The auto-reconnect support option can be activated regardless of the reliability mode.

Multi-threading Support

The Non-blocking API supports an unlimited number of threads calling its methods simultaneously.

Note In a multi-threaded scenario for the Non-blocking API, the order of invocation is **guaranteed**. The API performs operations in the same chronological order that they were called.

ResultHandler Interface

The Non-blocking API enables setting a result handler. A result handler is an interface with two methods, `handleSuccess` and `handleError`, as outlined in the following code:

```
public interface ResultHandler {

   /**
    * handle a successful result
    */
   public void handleSuccess(long handle, Object result);

   /**
    * handle a failure result
    */
   public void handleError(long handle, Object result);
}
```

You should implement this interface if you want to be informed about the success/error results of operations performed through the API.

Note This is the **only** interface for retrieving results; they **cannot** be returned immediately after the API method has returned to the caller.
In order to be able to receive operation results, you should set the result handler of the API before calling API methods whose results you want to receive. It is a good practice to set the result handler after the API is connected (as in the example below).

Both `handleSuccess` and `handleError` methods accept two parameters:

- *`Handle`*—Each API operation's return-value is a handle of type `long`. This handle enables correlation between operation calls and their results. When a `handle...` operation is called with a handle of value *X*, the result will match the operation that returned the same handle value (*X*) to the caller.
- *`Result`*—The actual result of the operation. Some operations may return a result of NULL.

Example

The following example is a simple implementation of a result handler that prints a message to `stdout` (when the result is successful) or to **stderr** (when the result is failure). This main method initiates the API and assigns a result handler.

For correct operation of the result handler, follow the code sequence given in this example.

Note This example does **not** demonstrate the use of callback handles.

```
import com.pcube.management.framework.rpc.ResultHandler;
import com.pcube.management.api.SMNonBlockingApi;


public class ResultHandlerExample implements ResultHandler{

    public void handleSuccess(long handle, Object result) {
        System.out.println("success: handle="+handle+
                                        ", result="+result);
    }

    public void handleError(long handle, Object result) {
        System.err.println("error: handle="+handle+
                                        ", result="+result);
    }

    public static void main (String args[]) throws Exception{
        if (args.length != 1) {
            System.err.println
                                       ("usage: ResultHandlerExample <sm-
ip>");
            System.exit(1);
        }

        //note the order of operations!
        SMNonBlockingApi nbapi = new SMNonBlockingApi();
        nbapi.connect(args[0]);
        nbapi.setResultHandler(new ResultHandlerExample());
        nbapi.login(...);
    }
}
```

Non-blocking API Construction

In addition to the constructors described in *API Construction* (on page 2-2), the Non-blocking API provides constructors that enable setting the reconnect period and the reliability mode.

Syntax

The syntax for the additional Non-blocking API constructors is shown in the following code block:

```
public SMNonBlockingApi(long autoReconnectInterval)

public SMNonBlockingApi(boolean reliable, long autoReconnectInterval)

public SMNonBlockingApi(String legName, long autoReconnectInterval)

public SMNonBlockingApi(String legName,
   boolean reliable, long autoReconnectInterval)
```

Arguments

The following is a description of the constructor arguments for the additional Non-blocking API constructors:

- *autoReconnectInterval*

Defines the interval (in milliseconds) for attempting reconnection by the reconnection task, as follows:

- If the value is 0 or less, the reconnection task is not activated (no auto-reconnect is attempted).
- If the value is greater than 0 and if there is a connection failure, the reconnection task will be activated every *`autoReconnectInterval`* milliseconds.

The default value is -1 (no auto-reconnect is attempted).

Note To enable the auto-reconnect support, the `connect` method of the API **must** be activated at least once. For more information see, *Non-blocking API Code Examples* (on page 4-9).

- *`reliable`*

  A flag that defines whether the API should work in reliable mode, as follows:

  - TRUE—the API works in reliable mode.
  - FALSE—the API works in non-reliable mode.

  The default value is TRUE (the API works in reliable mode).

- *`legName`*

  The name of the LEG, as described in *API Construction* (on page 2-2).

Examples

The following code constructs a reliable API with an auto-reconnection interval of 10 seconds:

```
SMNonBlockingAPI nbapi = SMNonBlockingAPI(10000);
nbapi.connect(<SM IP address>);
```

The following code constructs a reliable API without auto-reconnection support:

```
// API construction
SMNonBlockingAPI nbapi = SMNonBlockingAPI();

// Connect to the API
nbapi.connect(<SM IP address>);
```

The following code constructs a non-reliable API with auto-reconnection support:

```
// API construction
SMNonBlockingAPI nbapi = SMNonBlockingAPI(false,10000);

// Initial connection - to enable the reconnect task
nbapi.connect(<SM IP address>);
```

Non-blocking API Initialization

The Non-blocking API enables initializing certain internal properties for API customization. This initialization is performed using the API `init` method.

Note For the settings to take effect, the `init` method must be called **before** the `connect` method.

The following properties can be set:

- Output queue size—the internal buffer size defining the maximum number of requests that can be accumulated by the API until they are sent to the SM. The default is 1024.
- Operation timeout—the desired timeout (in milliseconds) on a non-responding PRPC protocol connection. The default is 45 seconds.

Syntax

The syntax for the Non-blocking API `init` method is as follows:

```
public void init(Properties properties)
```

Parameters

The following is a description of the parameters for the Non-blocking API `init` method:

- *`properties`* `(java.util.Properties)`

  Enables setting the following properties described above:

  - To set the output queue size, use `prpc.client.output.machinemode.recordnum`
  - To set the operation timeout, use `prpc.client.operation.timeout`

Example

The following code illustrates how to customize properties during initialization when using the Non-blocking API. Note that the `init` method is called **before** the `connect` method.

```
// API construction
SMNonBlockingAPI nbapi = SMNonBlockingAPI(10000);

// API initialization
java.util.Properties p = new java.util.Properties();
p.setProperty("prpc.client.output.machinemode.recordnum", 2048);
p.setProperty("prpc.client.operation.timeout", 60000);          // 1
minute
nbapi.init(p);

// initial connect to the API to enable the reconnect task
nbapi.connect(<SM API address>);
```

Non-blocking API Methods

This section describes the methods of the Non-blocking API.

All methods return a handle of type *long* that can be used to correlate operation calls and their results. See the *ResultHandler Interface* (on page 4-3) section.

The operation results passed to the result handler are similar to the return values described in the same method in the *Blocking API* (on page 3-1), with the exception of:

- Basic types are converted to their Java class representation. For example, `int` is translated to `java.lang.Integer`.
- Return values of `Void` are translated to `NULL`.

Note An error will be passed to the result handler **only if** the matching operation in the Blocking API throws an exception with the same arguments according to the SM database state at the time of the call.

All methods will throw a `java.lang.IllegalStateException` if called before a connection with the SM is established.

The following methods are described in this section:

- *login* (on page 4-7)
- *logoutByName* (on page A-1)
- *logoutByNameFromDomain* (on page A-1)
- *logoutByMapping* (on page A-1)
- *loginCable* (on page A-1)
- *logoutCable* (on page A-1)

login

Syntax

```
public long login(String subscriberName,
                  String[] mappings,
                  short[] mappingTypes,
                  String[] propertyKeys,
                  String[] propertyValues,
                  String domain,
                  boolean isMappingAdditive,
                  int autoLogoutTime)
```

The operation functionality is the same as the matching Blocking API operation.

logoutByName

Syntax

```
public long logoutByName(String subscriberName,
                         String[] mappings,
```

```
                                    short[] mappingTypes)
```

The operation functionality is the same as the matching Blocking API operation.

logoutByNameFromDomain

Syntax

```
public long logoutByNameFromDomain(String subscriberName,
                                   String[] mappings,
                                   short[] mappingTypes,
                                   String domain)
```

The operation functionality is the same as the matching Blocking API operation.

logoutByMapping

Syntax

```
public long logoutByMapping(String mapping,
                            short mappingType,
                            String domain)
```

The operation functionality is the same as the matching Blocking API operation.

loginCable

Syntax

```
public long loginCable(String CPE,
                       String CM,
                       String IP,
                       int lease,
                       String domain,
                       String[] propertyKeys,
                       String[] propertyValues)
```

The operation functionality is the same as the matching Blocking API operation.

logoutCable

Syntax

```
public long logoutCable(String CPE,
                        String CM,
                        String IP,
                        String domain)
```

The operation functionality is the same as the matching Blocking API operation.

Non-blocking API Code Examples

This section illustrates a code example for logging in and logging out subscribers.

Login and Logout

The following example logs in a predefined number of subscribers to the SM and then logs them out. Note the implementation of a *disconnect listener* and a *result handler*.

```
package nonblocking;

import com.pcube.management.framework.rpc.DisconnectListener;
import com.pcube.management.framework.rpc.ResultHandler;
import com.pcube.management.api.SMNonBlockingApi;
import com.pcube.management.api.SMApiConstants;

class LoginLogoutDisconnectListener implements  DisconnectListener {
    public void connectionIsDown() {
        System.err.println("disconnect listener:: connection is down");
    }
}

class LoginLogoutResultHandler implements ResultHandler {
    int count = 0;

    //prints a success result every 100 results
    public synchronized void handleSuccess(long handle, Object result) {
        Object tmp = null;
        if (++count%100 == 0) {
            tmp = result instanceof Object[] ?
                ((Object[])result)[0] : result;
            System.out.println("\tresult "+count+":\t"+tmp);
        }

    }
    //prints every error that occurs
    public synchronized void handleError(long handle, Object result) {
        System.err.println("\terror: "+count+":\t"+ result);
        ++count;
    }

    //waits for result number 'last result' to arrive
    public synchronized void waitForLastResult(int lastResult) {
        while (count<lastResult) {
            try {
                wait(100);
            } catch (InterruptedException ie) {
                ie.printStackTrace();
            }
        }
    }
}


public class LoginLogout {
    public static void main (String args[]) throws Exception{
        //check arguments
        checkArguments(args);
        int numSubscribersToLogin = Integer.parseInt(args[2]);
```

```
        //instantiation
        SMNonBlockingApi nbapi = new SMNonBlockingApi();
        try {
            //initation
            nbapi.setDisconnectListener(
                new LoginLogoutDisconnectListener());
            nbapi.connect(args[0]);
            LoginLogoutResultHandler resultHandler =
            new LoginLogoutResultHandler();
            nbapi.setResultHandler(resultHandler);
            //login
            System.out.println("login of "+numSubscribersToLogin
                               +" subscribers");
            for (int i=0; i<numSubscribersToLogin; i++) {
                nbapi.login("subscriber"+i,    //subscriber name
                            getMappings(i),    //a single ip mapping
                            new short[]{
                                SMApiConstants.MAPPING_TYPE_IP
                            },
                            null,          //no properties
                            null,
                            args[1],       //domain
                            false,         //mappings are not additive
                            -1);           //disable auto-logout
            }
            resultHandler.waitForLastResult(numSubscribersToLogin);
            //logout
            System.out.println("logout of "+numSubscribersToLogin
                               +" subscribers");
            for (int i=0; i<numSubscribersToLogin; i++) {
                nbapi.logoutByMapping(getMappings(i)[0],
                                      SMApiConstants.MAPPING_TYPE_IP,
                                      args[1]);
            }
            resultHandler.waitForLastResult(numSubscribersToLogin*2);
        } finally {
            nbapi.disconnect();
        }
    }
    static void checkArguments(String[] args)  throws Exception{
        if (args.length != 3) {
            System.err.println("usage: java LoginLogout "+
                               "<SM-address> <domain> <num-susbcribers>");
            System.exit(1);
        }
    }
    //'automatic' mapping generator
    private static String[] getMappings(int i) {
        return new String[]{ "10." +((int)i/65536)%256 + "." +
            ((int)(i/256))%256 + "." + (i%256)};
    }
}
```

APPENDIX A

List of Error Codes

Error codes are used for interpreting the actual error for which an `RpcErrorException` was returned. The error code is extracted using the `getErrorCode` method.

The error code enumeration is given in the `com.pcube.management.api.SMApiConstants` interface. A list of the error codes and their descriptions are given in the following table.

Table A-1 List of Error Codes

Error Code	Description
ERROR_CODE_BAD_SUBSCRIBER_MAPPING	A mapping was formatted badly or assigned to the subscriber illegally.
ERROR_CODE_DOMAIN_NOT_FOUND	The domain provided to the operation does not exist in the SM domain repository.
ERROR_CODE_ILLEGAL_ARGUMENT	One of the arguments provided to the method is illegal.
ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME	The subscriber name provided has more than 40 characters or has illegal characters.
ERROR_CODE_NOT_A_SUSBCRIBER_DOMAIN	The domain provided to the operation exists in the SM domain repository but is not a subscriber domain.
ERROR_CODE_NUMBER_FORMAT	A VLAN mapping string provided to the API does not represent a decimal number.
ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST	The subscriber on which the operation is performed does not exist in the SM database.
ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION	The subscriber exists in the SM database but is associated with a domain other than the one specified by the operation.
ERROR_CODE_SUBSCRIBER_MAPPING_CONGESTION	The mappings provided for the subscriber by the operation already belong to another subscriber.
ERROR_CODE_SUSBSCRIBER_ALREADY_EXISTS	The subscriber on which the operation was performed already exists in the SM database.
ERROR_CODE_DATABASE_EXCEPTION	Internal SM error – database error occurred during the operation.

Error Code	Description
ERROR_CODE_ARRAY_ACCESS	Internal SM error.
ERROR_CODE_ATTRIBUTE_NOT_FOUND	Internal SM error.
ERROR_CODE_CLASS_CAST	Internal SM error.
ERROR_CODE_CLASS_NOT_FOUND	Internal SM error.
ERROR_CODE_CLIENT_INTERNAL_ERROR	Internal error.
ERROR_CODE_CLIENT_OUT_OF_THREADS	Internal error.
ERROR_CODE_ILLEGAL_STATE	Internal SM error.
ERROR_CODE_OBJECT_NOT_FOUND	Internal SM error.
ERROR_CODE_OPERATION_NOT_FOUND	Internal SM error.
ERROR_CODE_OUT_OF_MEMORY	Internal SM error.
ERROR_CODE_RUNTIME	Internal SM error.
ERROR_CODE_NULL_POINTER	Internal SM error.
ERROR_CODE_SE_ERROR	Internal SM error. The SM could not perform the operation on the SCE device.
ERROR_CODE_UNKNOWN	Internal SM or API error.

Index